Paper category: Genetic Programming (submitted to GP-98)

Learning Nonlinear Predictive Models for Lossless Image Compression

Alex Fukunaga and Darren Mutz

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr., M/S 525-3660
Pasadena, CA 91109-8099
alex.fukunaga@jpl.nasa.gov, darren.mutz@jpl.nasa.gov
(626)306-6157

Abstract

We describe a genetic programming system which learns nonlinear predictive models for lossless image compression. S-expressions which represent nonlinear predictive models are learned, and the error image is compressed using an adaptive Huffman encoder. We show that the proposed system is capable of achieving compression ratios superior to that of the best known lossless compression algorithms.

1 Introduction

Lossless image compression is a problem with many real-world applications which has been studied by many researchers. The current dominant paradigm for lossless compression is predictive coding. State of the art lossless image compression algorithms (based on predictive coding) include the CALIC algorithm of Wu and Memon [15] and the LOCO-I algorithm of Weinberger et al. [14]. Reviews of lossless image compression can be found in [4, 5].

This paper proposes the use of genetic programming (GP) [3] in lossless image compression as the mechanism for representing and learning nonlinear models for predictive coding. Because of the enormous computational cost of evolving nonlinear predictive models would be prohibitively expensive using standard GP systems, we have implemented a highly efficient, *genome-compiler* GP system which compiles s-expressions into native machine code to enable the application of GP to this problem.

We evaluate our GP-based compression system by comparison with the state of the art lossless image compression algorithms and show that it is possible to obtain compression ratios superior to the best known algorithms.

The rest of the paper is organized as follows. In Section 5, we review predictive coding based image compression. Section 3 describes the GP-based compression system. Section 4 presents an empirical evaluation of our system using some test images, and compares the results with that of the best known lossless compression algorithms. We discuss related work in 5, and we conclude in Section 6 with a discussion and directions for future work.

2 Predictive Coding Based Image Compression

Predictive coding is an image compression technique which uses a compact model of an image to predict pixel values of an image based on the values of neighboring pixels. A *model* of an image is a function

```
Encoder(Model,Image)
  for x = 0 to xmax
    for y = 0 to ymax
      Error[x,y] = Image[x,y] - Model(x,y)
Decoder(Model)
  for x = 0 to xmax
    for y = 0 to ymax
      Image[x,y] = Model(x,y) + Error[x,y]
```

Figure 1: Algorithm schema for predictive coding. $Model(x, y)$ is a function that takes the coordinates of a pixel and returns a predicted value of that pixel. *Image* and *Error* are two-dimensional arrays.

$model(x, y)$, which computes (predicts) the pixel value at coordinate (x, y) of an image, given the values of some *neighbors* of pixel (x, y), where neighbors are pixels whose values are known. Typically, when processing an image in raster scan order (left to right, top to bottom), neighbors are selected from the pixels above and to the left of the current pixel. For example, a common set of neighbors used for predictive coding is the set $\{(x\text{-}1,y\text{-}1), (x,y\text{-}1), (x+1,y\text{-}1),(x\text{-}1,y)\}$. *Linear predictive coding* is a simple, special case of predictive coding in which the model simply takes an average of the neighboring values. *Nonlinear* models assign arbitrarily complex functions to the models.

Suppose that we have a perfect model of an image, i.e., one which can perfectly reconstruct an image given the pixel value of the border pixels (assuming we process the pixels in raster order). Then, the value of the border pixels and this compact model is all that needs to be transmitted in order to transmit the whole information content of the image. In general, it is not possible to generate a compact, perfect model of an image, and the model generates an *error signal* (the differences at each pixel between the value predicted by the model and the actual value of the pixel in the original image.

There are two expected sources of compression in predictive coding based image compression (assuming that the predictive model is accurate enough). First, the error signal for each pixel should have a smaller magnitude than the corresponding pixel in the original image (therefore requiring fewer bits to transmit the error signal). Second, the error signal should have less entropy than the original message, since the model should remove of much of the "principal components" of the image signal.[1] To complete the compression, the error signal is compressed using an entropy coding algorithm such as Huffman coding or arithmetic coding [6]. State of the art algorithms such as CALIC also perform context modeling prior to applying entropy coding – see [4, 5]. Our system does not apply context modeling techniques.

If we transmit this compressed error signal as well as the model and all other peripheral information, then a receiver can reconstruct the original image by applying an analogous decoding procedure (see Figure 1).

3 Evolving Nonlinear Predictive Models with a GP

Given an image, we use genetic programming to generate a Lisp s-expression which represents a nonlinear model of the image predictive coding based compression.

The terminals used for genetic programming were:

- values of the four neighboring pixels *Image[x-1,y-1], Image[x,y-1], Image[x+1,y-1], Image[x-1,y].*
- selected constant values: 1, 5, 10, 100.

The functions used were:

[1] If the model were perfect, then the error signals would consist of all 0's, and can be compressed to a single byte.

- arithmetic functions: +,-,*,% (protected division [3])
- MIN(a,b) and MAX(a,b) functions which return the minimum and maximum values of their two arguments, respectively.

As we noted in Section 2, a standard entropy coding algorithm needs to be applied to the error image. For this experiment, we used an adaptive Huffman coder[2] as the entropy coder. In addition, note that given the four pixel neighborhood we use, the pixel values of the borders of the image, i.e., the top row, the leftmost column, and the rightmost column need to be stored directly (these are the border cases for which we can not apply the predictive model). Also, the model (which is unique for each image) must also be stored in the compressed image data. We applied Unix *compress* (which uses Lempel-Ziv coding) to the border pixels and the model, and concatenated these to the Huffman-coded error signal. Finally, two integer values indicating the size of the image (height,width) were added to the file. Given this data, we can reconstruct an image without loss of information.

Thus, the exact file size of the compressed image (the values reported in the experiments below) is:
sizeof(HuffmanCodedError) + sizeof(HuffmanCodeBook)+ sizeof(CompressedBorder) + sizeof(CompressedModel) + sizeof(2 integers)

In the experiments described below, our GP system was configured as follows: population=500, generations=30, tournament selection (size=5), 90% crossover, 10% reproduction, no mutation).

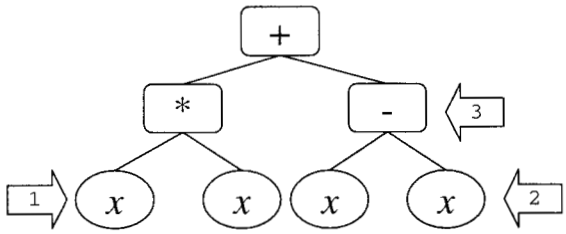
Because the s-expression models are unconstrained in the range of values that they can return, the worst-case size of the codebook is actually the number of pixels in the image, even though there are only 256 unique pixel values in the original image. This means that the size of the codebook can possibly become a very significant component of the compressed data that must be transmitted. We therefore implemented a execution speedup heuristic which abandons evaluation of an individual once the number of unique error image pixel values generated exceeds either 40000 or 25% of the total number of pixels in the input image. By doing this, we avoid the cost of running the entropy coder for individuals that both a) look very unpromising and b) will be extremely computationally expensive to evaluate.

3.1 Genome Compiler

In order to evaluate a single individual, the s-expression needs to be evaluated for each pixel in the image (excluding the borders). This requires hundreds of thousands to millions of repeated execution of the same s-expression per evaluation. We originally implemented the image compression application based on lil-gp 1.1 [9], a well-known, efficient C implementation of GP This implementation required almost two weeks of CPU time to execute a 50 generation, population 2000 GP run for compressing a 256 by 256 image on a 296-MHz Ultrasparc 2. Not only was this much too slow for practical use, it made experimentation infeasible. Thus, we sought to make individual evaluations as efficient as possible.

We therefore extended lil-gp by implementing a *genome compiler* which translates s-expressions into efficient SPARC machine code prior to execution. The major benefit of compilation is the removal of function call overhead during the s-expression tree evaluation, which we found was responsible for the vast majority of the computation time in the standard lil-gp based system. While the compilation process itself requires some computational overhead, we found that this was negligible when the same s-expression was evaluated many times. Using the current implementation of our genome compiler, we can execute a 50 generation, population 2000 GP run for compressing a 256 by 256 image in about 2-3 days on an 296MHz Ultrasparc 2, which is significantly better than the 10-15 days it took prior to the implementation of the compiler. Figure 2 shows an example of this compilation process.

[2]More specifically, we used the Huffman coder in [?] and computed the character frequencies before initializing the coder.




```
[1] mov  fp31, fp0
    mov  fp31, fp1
    mult fp0, fp1, fp0
    mov  fp31, fp1
    mov  fp31, fp2
[2] sub  fp1, fp2, fp1
[3] add  fp0, fp1, fp0
```

Figure 2: Example of s-expression compilation: An s-expression with corresponding assembly-level code. Numbered breaks in the code correspond to the code generated so far when the post order traversal has progressed to the node indicated in the tree diagram.

The genome compiler's performance, [3] when applied to problems where individuals are repeated many times, compares favorably with the fastest reported GP systems, including the CGPS system of Nordin and Banzhaf [7], which directly manipulates SPARC machine code, yielding roughly two orders of magnitude speedup over lil-gp on symbolic regression problems.[4] Further details about the genome compiler will be available in a forthcoming report.

4 Results

The genetic programming system for evolving models for predictive coding lossless image compression was evaluated by comparing the size of the compressed files with a number of standard lossless compression algorithms on a set of grey scale images. The images used were science images of planetary surfaces taken from the NASA Galileo Mission image archives (we used these images because these are of greatest interest to our sponsor). Note that in this report, we focus on grey scale images, but the technique can be straightforwardly extended to color images by operating on three image planes (red,

[3] By which we mean the comparative speed in time to execute a GP run over standard C implementations of GP such as lil-gp[9] and SGPC [13] running on the same problem the same machine as the genome compiler.

[4] Note that for image compression, the majority of the time is now spent in the adaptive Huffman coder, which is why the speedup compared to the standard lil-gp implementation is only around a factor of 5. This point is further discussed in 6.

blue, green) – this is what many state of the art algorithms such as CALIC do.

The compression ratio of the following algorithms are shown in Table 1.

- *evolved*: The evolved predictive coding compression algorithm
- *CALIC*: A state of the art lossless image compression algorithm, described in [15]. In general, this algorithm provides the best compression ratio among previous algorithms.
- *LOCO-I*: This is an algorithm developed by Weinberger et al [14] which was recently selected as the new ISO JPEG-LS (lossless JPEG) baseline standard.
- *gzip, compress, pack*: These are standard Unix string compression utilities; *gzip* implements the Lempel-Ziv (LZ77) algorithm, *compress* implementes the adaptive Lempel-Ziv-Welch (LZW) algorithm, and *pack* uses Huffman coding.
- *szip*: A software simulation of the Rice Chip, the current standard lossless compression hardware used by NASA.

It important to note that a different model is evolved for each image that the genetic programming system is applied to. In contrast, the other approaches (CALIC, GIF, etc.) apply a single model to every image. Thus, the time to compress an image using the genetic programming approach is several orders of magnitude greater than the time it takes to compress an image using other methods. However, the time to decompress an image is competitive with other methods. Therefore, the genetic programming system is an example of an *asymmetric compression algorithm* (slow compression, fast decompression).

Image name	original size	evolved	CALIC	LOCO-I (JPEG-LS)	compress	gzip	pack	szip
earth	72643	30380	31798	32932	42502	40908	55068	40585 (30)
earth4-128	11246	5513	5631	5857	7441	6865	8072	7727 (30)
earth6	20400	9288	10144	10488	11339	10925	13264	12793 (8)
earth7	21039	10218	11183	11476	13117	12520	15551	13269 (18)
earth8	19055	9594	10460	10716	11699	11350	13298	12465 (20)

Table 1: Compression ratios of various compression techniques applied to set of test images.

As Table 1 shows, the compressed file sizes obtained using the GP-evolved models is superior to all of the other algorithms for these test images. Due to the extremely long runtimes for evolving compressed images (at the time that the experiments were being run, the genome compiler was still under development), we have only collected data for a small number of runs. We are currently in the process of collecting more data to better understand the relative efficacy of evolutionary compression on a wide range of image classes.

5 Related Work

Salami developed part of an evolvable hardware system for lossless image compression [10]. They used a genetic algorithm to evolve weights for a linear predictor for predictive coding, and showed that the entropy of the error image for some test images was lower than that of CALIC and LOCO (the error image entropy was measured instead of the compression ratio, since they did not implement an entropy coder). However, note that the entropy of the error image is *not* necessarily indicative of the relative compression ratio obtained by a complete compression system, and is only a first-order approximation for actual compression results. This is in part because entropy coding techniques such as Huffman Coding do not perform optimally for arbitrary distributions of error pixel values, and actual performance of the coders depends largely on the error pixel distributions. Furthermore, CALIC and LOCO apply context modeling techniques to the error image to further reduce entropy, and the error

image entropy metric used in [10] does not take this into account. For example, the error image entropy for the *barb* test image used by Salami et al is smaller for LOCO (5.625) than for CALIC (5.636), but the number of bits/pixel in the final compressed file is smaller for CALIC (4.41) than for LOCO (4.69). Also, the old standard JPEG[5] obtains the lowest error image entropy on two of Salami et al's images, even though the final compressed file size is the worst (by a significant margin) among the three algorithms for all eight of the images. These discrepancies between error image entropies and actual compression performance indicates that error image entropy is *not* a good metric for comparing different compression algorithms. Thus, while Salami et al's work was a very encouraging, important first step in evolving linear predictive coders, it is not clear that the low-entropy error images discovered by their system could be further encoded into a file that is smaller than the compressed files created by state of the art algorithms such as CALIC and LOCO.

Our work differs from [10] in that we evolved nonlinear predictive models using genetic programming, as opposed to linear models in evolvable hardware. Furthermore, we have implemented a complete compression system, and showed that this approach is capable of yielding smaller compressed files than that of the best known lossless compression algorithms, CALIC and LOCO. However, because our system is implemented entirely in software on general purpose hardware, it is likely to be significantly slower than the speeds achievable using a dedicated evolvable hardware platform.

Neural networks have previously been used to learn nonlinear predictors (c.f. [1]). However, it is difficult to compare these approaches with our work because they do not compare their system to current, standard lossless image compression algorithms (this is in part because many of the advances in lossless compression are quite recent [4, 5].

Salami et al. have also developed an evolvable hardware system for *lossy* compression which evolves nonlinear predictive models on function level evolvable hardware [12, 11].

Image compression using GP has been previously studied in the context of *programmatic compression*, where essentially, an image is used as the target function for symbolic regression. Koza [3] initially demonstrated this technique on a 30 by 30 bitmap. Nordin and Banzhaf [8] used CGPS, a very efficient GP system, and heuristics such as *chunking* (tiling an image into smaller subimages which were separately compressed) to scale up the technique to 256 by 256 images. Programmatic compression is theoretically capable of being lossless (if a perfect model of the image is discovered), but it is essentially a *lossy* technique, and is very different from lossless predictive coding compression.

Work in the implementation of high-performance GP systems closely related to our genome compiler includes that of Nordin and Banzhaf, [7], whose CGPS system directly manipulates SPARC machine code, and Juille and Pollack [2], whose system compiles s-expressions into virtual stack-based machine code.

6 Discussion and Future Work

The research reported here is preliminary, and is only the first step in understanding the capabilities and limitations of using genetic programming for lossless image compression.

We believe that our initial results are quite promising, since they show that the evolved models can yield an improvement over CALIC, which is currently the best known lossless image compression algorithm. Furthermore, these results were obtained without any special tuning of algorithm control parameters or the function/terminal sets for the GP system.

However, it should be noted that the genetic programming system takes several orders of magnitude more time to evolve a model that achieves its superior results (several hours per image) than the other approaches (which run in a few seconds). Although slow compression times are acceptable for some applications, as long as decompression is fast (e.g., for archiving images), this is not acceptable for applications requiring near real-time compression times. In addition, the slowness of the current GP system makes it difficult for us to experiment with variations on the technique, because it takes too much time to gather statistically significant data.

[5] This refers to the previous standard lossless JPEG algorithm, prior to the recent selection of the new LOCO-I/JPEG-LS standard.

We have implemented a genome compiler to speed up the execution time (see Section 3.1, and have succeeded in obtaining a factor of 5 speedup over standard GP for 256x256 images. However now the vast majority (98% for 256x256 images) of the execution time is spent in the adaptive Huffman coding code. Therefore, a promising avenue explore is to evolve models by using error image entropy as the fitness measure, instead of compression ratio (as done by [10]) – this would allow us to avoid running the Huffman encoder for each individual evaluation, resulting in significant execution speedup. However, this may result in degraded compression performance, since, as we noted above, error image entropy is not always a reliable predictor of compression ratio. A compromise would be to develop a modified objective function computation scheme in which we only execute the Huffman coder for individuals that generate error image entropies which are not significantly worse than that of the best individual found so far. This would enable us to eliminate entropy coding for the majority of individual evaluations, and will likely lead to an order of magnitude speedup.

There are many more directions which can be pursued in evolutionary image compression. First, it seems worthwhile to experiment with different entropy coders. Second, there are more sophisticated predictive coding architectures which can be explored. One such example would be a two-pass model in which a standard (evolved or hand-coded) linear model is first applied to minimize the first order entropy, then followed by the application of an evolved nonlinear model to model the error after the linear model is applied. Third, we can divide the image[6] into subimages and apply the compression separately to the subimages.

Finally, a very intriguing prospect would be to try to evolve models which are more general than the image-specific nonlinear models that were explored in this paper. By using sets of images as fitness cases instead of a single image, it is possible to try to evolve a single predictive model (such as those handcoded in traditional lossless compression algorithms) which works well for a large class of images. This will no doubt require an enormous amount of computation. However, it may be possible to generate a nonlinear model, which when possibly combined with context modeling techniques, yields a better, general purpose model than those used by state of the art algorithms.

7 Acknowledgments

The research described in this paper was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. Thanks to Adrian Stoica, Benny Toomarian, and Ken Hayworth for helpful discussions, and to Bill Punch and Douglas Zonker for making lil-gp publically available.

References

[1] W.W. Jiang, S.-Z Kiang, N.Z. Hakim, and H.E. Meadows. Lossless compression for medical imageing systems using linear/nonlinear prediction and arithmetic coding. In *Proc. IEEE International Symposium on Circuits and Systems*, volume 1, pages 283–6, 1993.

[2] H. Juille and J.B. Pollack. Massively parallel genetic programming. In P. Angeline and K. Kinnear, editors, *Advances in Genetic Programming 2*. MIT Press, 1996.

[3] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[4] N. Memon and X. Wu. Lossless compression. In *CRC Handbook of Communication*. 1996 (to appear).

[5] N. Memon and X. Wu. Recent progress in lossless image coding. *The Computer Journal, to appear*, 1997.

[6] M. Nelson and J-L. Gailly. *The Data Compression Book (second edition)*. M&T Books, 1996.

[7] P. Nordin and W. Banzhaf. Evolving turing-complete programs for a register machine with self-modifying code. In *Proceedings of the International Conference on Genetic Algorithms*. Morgan Kaufmann, 1995.

[6] Nordin and Banzhaf called this "chunking" [8]; Salami et al [11] have applied this technique in evolvable hardware based lossy compression

[8] P. Nordin and W. Banzhaf. Programmatic compression of images and sound. In *Proceedings of the Annual Genetic Programming Conference*, pages 345–350, 1996.

[9] B. Punch and D. Zonker. lil-gp genetic programming system version 1.1 beta version. Michigan State University, http://GARAGe.cps.msu.edu/software/lil-gp/index.html, 1996.

[10] M. Salami, M. Iwata, and T. Higuchi. Lossless image compression by evolvable hardware. In *Proc. European Conf. on Artificial Life*. MIT Press, 1997.

[11] M. Salami, M. Murakawa, and T. Higuchi. Data compression based on evolvable hardware. In *International Conference on Evolvable Systems*. Springer Verlag LNCS, 1996.

[12] M. Salami, M. Murakawa, and T. Higuchi. Lossy image compression by evolvable hardware. In *Proc. Evolvable Systems Workshop, International Joint Conference on Artificial Intelligence*, 1997.

[13] W. Tackett and A. Carmi. sgpc: simple genetic programming in c. ftp://ftp.io.com/pub/genetic-programming, 1993.

[14] M.J. Weinberger, G. Seroussi, and G. Sapiro. Loco-i: A low complexity, context-based, lossless image compression algorithm. In *Proceedings of the Data Compression Conference (DCC'96)*, pages 140–149, 1996.

[15] X. Wu and N. Memon. Context-based, adaptive, lossless image codes. *IEEE Transactions on Communications*, 45(4), 1997.